GATEWAY FOR SERVICE ORIENTED STATE

BACKGROUND

[0001] Today there is no built in support in a service-oriented architecture (like web services) to inform a client of its state changes. This is an important feature for a service-oriented architecture as most computing systems are asynchronous in nature and the client needs to acquire such a change of the state through some asynchronous mechanisms.

[0002] In a service oriented component model (java services and/or web services), the state of a component is held by the service through its instance variables and/or session state variables and/or persistence storage media and/or using external resources. The service exposes its state information through various custom application programming interfaces (APIs). This interface model is not flexible and extensible, therefore each client has to program code to fit-in specific APIs and to understand the semantics as defined through the APIs. The clients are not flexible enough to query state data based on its exposed information model (data schema). The granularity and flexibility of these APIs vary with different implementation.

[0003] Therefore, what is needed is a framework to define mechanisms that enable a service to extend its state with change notification semantics a flexible service state exposure model through various queries on the state of the service using the exposed state information model (schema).

SUMMARY OF THE INVENTION

[0004] These and other improvements are set forth in the following detailed description. For a better understanding of the invention with advantages and features, refer to the description and to the drawings.

[0005] Disclosed herein in an exemplary embodiment is a method for managing state data of a service in a service-oriented architecture by establishing a gateway for service-oriented state comprising: configuring an extensible, pluggable interface to support for extensible processor interfaces; data query support on service state data, automated notification capability on service state to a client; and automated data transform on service state data to a client format; defining an interface framework for interaction between a service and the gateway. The method also includes establishing an extensible meta-data definition comprising an extensible set of service state data attributes including state data qualifiers, constraints, and access mechanisms; and utilizing one or more pluggable processors configured to utilize the extensible meta-data definition for interfaces and decision making based on the meta-data.

[0006] Also disclosed herein in another exemplary embodiment is a system for managing state data of a service in a service-oriented architecture by establishing a gateway for service-oriented state comprising: a means for configuring an extensible, pluggable interface to support for extensible processor interfaces; data query support on service state data, automated notification capability on service state to a client; and automated data transform on service state data to a client format; defining an interface framework for interaction between a service and the gateway. The system also includes a means for establishing an extensible meta-data definition comprising an extensible set of service state data attributes including state data qualifiers, constraints, and access mechanisms; and a means for utilizing one or more pluggable processors configured to utilize the extensible meta-data definition for interfaces and decision making based on the meta-data.

[0007] Disclosed in yet another exemplary embodiment is a storage medium encoded with a machine-readable computer program code, the code including instructions for causing a computer to implement a method for managing state data of a service in a service-oriented architecture by establishing a gateway for service-oriented state, the method comprising: configuring an extensible, pluggable interface to support for extensible processor interfaces; data query support on service state data, automated notification capability on service state to a client; and automated data transform on service state data to a client format; defining an interface framework for interaction between a service and the gateway. The storage medium also includes code for implementing the method including: establishing an extensible meta-data definition comprising an extensible set of service state data attributes including state data qualifiers, constraints, and access mechanisms; and utilizing one or more pluggable processors configured to utilize the extensible meta-data definition for interfaces and decision making based on the meta-data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The present invention will now be described, by way of an example, with references to the accompanying drawings, wherein like elements are numbered alike in the several figures in which:

[0009] FIGURE 1A depicts a simplified block diagram depicting and exemplary computer implementation;

[0010] FIGURE 1B depicts a simplified block diagram depicting client service interaction for a query;

[0011] FIGURE 1C depicts a simplified block diagram depicting client service interaction for notification of a state change;

[0012] FIGURE 2 illustrates a simplified block diagram of an information communication framework including a gateway for service oriented state in accordance with an exemplary embodiment;

[0013] FIGURE 3 illustrates a simplified block diagram of an information communication framework including a service oriented state generator in accordance with an exemplary embodiment;

[0014] FIGURE 4 depicts a relationship model in accordance with an exemplary embodiment; and

[0015] FIGURE 5 illustrates a simplified block diagram of an an information communication framework including a pluggable extensible meta data processor in accordance with an exemplary embodiment.

[0016] Our detailed description explains the preferred embodiments of our invention, together with advantages and features, by way of example with reference to the drawings.

DETAILED DESCRIPTION OF THE INVENTION

[0017] Referring to Figures 1A-C and 2, disclosed herein in an exemplary embodiment, is a method and apparatus for managing the state data of a service in a service-oriented architecture. More particularly, a framework 10a to define mechanisms that enable a service 20 to extend its state with change notification semantics. In an exemplary embodiment, a service 20 may reside in a computer 1 configured to communicate with various system elements aand media 2 including another computer with a client 30. Moreover, the client could reside in the same computer 1 as the service 20.

[0018] The framework 10a enables a service 20 component with various extensible data query support on service state data 22, automatic notifying capability on service state data 22 change and data transform flexibility on service state data 22. This framework 10a is flexible as clients 30 can write any query expressions on the service state data 22; for example, a high level view of the state data 22, or it may be a granular (fine-grain) view on the state data 22. Advantageously, this framework 10a is extensible as new type of service state can be plugged in to the service oriented architecture. In an exemplary embodiment, these features are achieved through pluggable processors 108, 110, 112 utilizing the extensible meta-data framework 10a related to state data qualifiers, constraints and access mechanism.

[0019] It will be appreciated that as used herein, *architecture* is intended to mean both the process and the specifying of the overall structure, logical components, and the logical interrelationships of a computer, its operating system, a network, or other conception. An architecture can be a *reference model*, such as the Open Systems Interconnection (OSI) reference model, intended as a model for specific product architectures or it can be a specific product architecture, such as that for a particular processor or operating system.

[0020] Computer architecture can be divided into five fundamental components: input/output, storage, communication, control, and processing. In practice, each of these components (sometimes called *subsystems*) is sometimes said to have an architecture, so, as usual, context contributes to usage and meaning.

[0021] By comparison, the term *design* connotes thinking that has less scope than architecture. An *architecture* is a *design*, but most designs are not architectures. A single *component* or a new *function* has a *design* that has to fit within the overall *architecture*.

[0022] A similar term, *framework*, may be thought of as the structural part of an architecture but as used herein may also include processes. A framework may include both the process and the specifying of the overall structure, logical components, and the logical interrelationships of a system, processor and the like.

[0023] A *tool set* may be thought of as a collection of tools or functions to create artifacts (e.g., effects, code, schema, data, and the like) from a set of iputs with the assistance of a user intervention.

[0024] A *component* is a physical or non-physical abstraction element that provides/facilitates a selected set of functionalities/capabilities of the architecture and/or framework.

[0025] Another beneficial feature of this framework 10a is the support of data transformation for the client 30. There are different types of clients, based on the heterogeneous nature of the transport, presentation media, QoS and formatting requirements. These clients 30 access the service 20 for its state data 22 and expect the results formulated/configured in a manner appropriate for their consumption. In an exemplary embodiment, a framework 10a for transformation of the service state data 22 for the client(s) 20 based on their requirements is provided. These requirements may include transformation of the service state data 22 to a format that they can understand and utilize; for example, a stock quote service state data from a stock quote service may be converted into Wireless Markup Language (WML) format for a mobile device display or into Voice Extensible Markup Lanuguage (XML) format for a voice enabled device.

[0026] It is noteworthy to appreciate that a service-oriented architecture and the framework 10a of an exemplary embodiment disclosed herein facilitate and provide the following:

1. Enable a query on a current state of the service 20, preferably using a client specified query language;
2. Enable the service 20 to send service state data 22 changes to selected interested clients.
3. Transform a result of such queries and notification information to a client specified format.

[0027] An exemplary embodiment addresses the above requirements for a service-oriented architecture with a set of meta-data information and pluggable/adaptable interfaces and tooling. Advantageously the meta-data framework 10a is extensible to support state data qualifiers (security, transaction enable/disable, transaction scope and the like), constraints (availability, time constraints, time to live, mutability, cardinality, etc) and access mechanisms (push, pull and custom access mechanisms).

[0028] Refering now to Figures 1B and Figure 1C, an illustration of the components involved in the framework 10a of an exemplary embodiment are depicted. An exemplary illustration of a process and communication between a service 20 and a client 30 are depicted. A service 20 e.g., Java Service / Web service, State Address Provider service, and the like, has some state information/data 22 such as an address, and would like to enable its clients 30 to query 40 this state using some query language including, but not limited to, *XQuery, ISQL, XSL EJBOL* and the like. For example the client 20 may query the zip code from the address. Moreover, as depicted in Figure 1C, a service 20, e.g., Java service, web service, State Address Provider service, has some state information, in this instance an Address and would like to send notifications 42 of a state change, e.g., address change, to the various clients 30 that are/may be interested in listening to the state data changes.

Gateway for a Service Oriented State

[0029] Turning now to Figure 2, a simplified block diagram of the framework 10a of an exemplary embodiment is depicted. In an exemplary embodiment the framework 10a includes and describes a Gateway for a Service Oriented State(GSOS) 100 and its features. A GSOS 100 is responsible for managing the service state data 22 in a service-oriented architecture. Service state data 22 may be a logical, or physical abstraction, state and/or state data 22 in logical sense could be just a pointer, or a physical construct, such as a real address. This service state data 22 abstraction and meta-data information may be created using the tools optionally provided along with this framework10a in another exemplary embodiment. This framework 10a employing the GSOS of an exemplary embodiment includes, but is not limited to, these components and functionalities:

1. A meta-data definition;
2. An extensible meta-data processor;
3. An interfaces framework to define the interactions between a service 20 and the GSOS 100.
4. A pluggable interface to support service state data Query, notification and transformation

Meta-data definition

[0030] The meta-data definition of an exemplary embodiment is an extensible set of state data constraints, qualifiers and data access mechanisms. The meta-data definition is an information model associated with the state data 22 and provides more semantic information on the service's 20 state. This meta-data definition may be an extension to the state data 22 schema through annotations on the schema or can be a more general-purpose language such as Resource Description Framework (RDF), DARPA Agent Markup Language(DAML), and others). In an exemplary embodiment the GSOS 100 employs an XML model to define this meta-data information.

An extensible meta-data processor

[0031] An extensible meta-data processor 300 (See also Figure 4) includes a rule engine to process the meta-data associated with the state data 22 of a service 20 . This rule engine is complex in nature by providing expert decision based on the meta-data. These decisions include when to notify a state change, interpret the semantics of the query and retrieve the state from an external resource (e.g., SQL, Simple Network Management Protocol (SNMP) server, Common Information Model Object Manager (CIMOM), web services) or from the service 20, make a decision on the transaction requirements and security constraints, to make decisions on the validity and availability of the state data 22 to make the decision about which query processor to use based on user and/or client query criteria. It will be appreciated that as described herein the extensible meta-data processor 300 is described as a part of or imbedded with the GSOS 100. However the meta-data processor 300 may also be pluggable with the GSOS providing the appropriate interfaces. It will be appreciated that the framework 10a provides an extension capability based on meta-data language extensibility to support custom scenarios.

Interface framework to define the interaction between a service and GSOS

[0032] In an exemplary embodiment, the GSOS 100 provides a standard interface to define the interactions between the service 20 and GSOS 100. In an exemplary embodiment three types of interface are employed:

A. State meta-data management interface 102;

B. Gateway access interface 104; and

C. State data access interface 106 from the service or from external resources.

A pluggable architecture to support service state data Query, notification and transformation

[0033] In an exemplary embodiment, the GSOS 100 also provides interface to a number of pluggable processors to facilitate and support service state data Queries, Client notification and transformations. It will be appreciated that while in an exemplary embodiment three such pluggable processors are depicted and described, any number of pluggable processors and interfaces are conceivable without loss of generality. Moreover, it should be further appreciated that it is also conceivable to employ multiple processors of similar types. For example, multiple query interfaces 108 and processors 110 may be utilized to facilitate the needs of the framework 10a, and/or client 30, and the like.

Query

[0034] The query interface 108 enables the GSOS 100 to support any query language and processor 110 of of a client's 20 choice. The user/client 20 creates a query based on the schema associated with the state. GSOS 100 is responsible for applying the query on the state data 22 based on the meta-data information and it presents the Query results on service state data 22 to the query processor 110 in a canonical schema format understandable to the processor 110. In addition GSOS 100 also provides caching mechanisms to support faster query process. In an illustrative implementation of an exemplary embodiment of a GSOS 100 and the query interface 108 supports an in-memory XML Document Object Model (DOM) programming interface.

Notification

[0035] The GSOS 100 supports pluggable notification processors 114 with a notification processor interface 112 based on selected notification requirements. These requirements include service state aggregation, queuing, periodic delivery, and asynchronous behaviors. GSOS 100, based on the service state meta-data information automatically sends the state change information to this processor 114 with the current and old value of the state data 22.

Transformation

[0036] Based on a client's 20 request, the GSOS 100 framework transforms the results of a query or change notification to a format as prescribed by the client 20. The GSOS 100 incluses a transfrom processor interface 116 to facilitate plug-in of any transformation processor 118 of choice.

SERVICE ORIENTED STATE DATA AND STATE META-DATA GENERATION USING META INFORMATION MODELING

[0037] Disclosed herein in another exemplary embodiment is a method and apparatus for generating service state data and extensible meta-data information based on the state data schema for a service-oriented architecture. This framework, uses a meta information model on the top of the state meta-data to provide more flexibility in modeling, versioning and compatibility. This enables the service component to be used with the framework utilizing the Gateway for Service Oriented State (GSOS) disclosed earlier, for various extensible data query support on service state data, automatic notifying capability on service state data change and data transform flexibility on service state data.

[0038] Referring now to Figure 3, a framework denoted here as 10b as an other exemplary embodiment and expansion of that described earlier, utilizing a service oriented state generator 200 hereinafter also denoted SSDG is depicted. In a service oriented component (e.g., java services and/or web services, enterprise, and the like), the state of a component is held by the service 20 through its instance variables and/or session state variables and/or persistence storage media and/or using external resources. The Gateway for Service Oriented State (GSOS) 100 and framework 10b, provides a flexible service state exposure model through various queries on the state of the service 20 using an exposed state information model (data schema). This framework 10b including the GSOS 100 is configured to be flexible and permits clients 20 to formulate query expressions on the service state data 22. This framework 10b also provides a means to define mechanisms to enable a service 20 to extend its state with change notification semantics. This extensible framework 10b depends on the state data and its meta-data information.

[0039] In an exemplary embodiment, a tool set shown generally as 12 (a *service state generator framework)* is defined to support creation of state data 22 with the necessary meta-data. Once again, this generated state data may be a logical or physical mapping to a service state data 22. This tool set 12 can operate on any data schema 210 (e.g., XML Schema, Document Type Definition (DTD), Regular Language Description For XML - Next Generaton (RELAX-NG) Schema). This service state generator 200 provides the maximum flexibility by using a meta-model 220 for modeling the state meta-data. The use of this modeling metaphor helps in defining a meta-data language with versioning, compatibility, a standard for tools to work on the state-data and a standard for code generation.

[0040] In an exemplary embodiment the tool set 12 and state data generator 200 provides numerous advantages and features for generation of service state data 22 utilizing various meta data and schema. For example, the tool set 12 and state data generator 200 generates state data, state data logical mapping and/or physical mapping, meta-data associated with the state data, and includes meta-data model 220 associated with the meta-data. The tool set 12 generates the necessary interfaces and code to support the state data and its usage in a service 20. In addition, this tool set 12 facilitates creation of the state data in the framework 10b as a logical grouping or as a direct physical representation with the necessary interfaces to support the meta-data requirements like serializes and deserializers, data push and pull mechanisms etc.

[0041] The tool set 12 and SSDG 200 works in conjunction with state data schema 210 and a meta-data language. The meta-data language is a common language for describing the meta-data through meta data models 220 (meta-data about meta-data) to facilitate and ensure compatibility with previous versions. This can be any existing XML languages or a derivative thereof. Moreover, the tool kit 12 facilitates extension to support new meta-data features. This toolset 12 works with state data schema 210; these state schema 210 may be associated with the WSDL (Web Service Description

Language) through port type (interface) extensibility or by including external schema definitions. The state data schema 210 may be, but are not limited to, XML schema, RELAX-NG, Reverse Description Framework (RDF), and the like.

[0042] Other inputs to the SSDG 200 may inlcude custom templates 230, meta-data attributes 240 and meta-information models etc.

a. Meta-data attributes 240 include, but are not limited to: Qualifiers: security, transaction, notifiability, and the like; constraints: state data mutability, Time To Live (TTL) features, cardinality, and relationship (workflow) information etc.; and Data access mechanisms: callback mechanisms, data push mechanisms, extensible data access mechanisms like connectivity to data base, Common Information Model Object TBD (CIMOM), Simple Network Management Protocol (SNMP) server and the like.

b. Meta information models 220 may be viewed as a schema or meta-data for the service state meta-data. Meta information modeling provides flexibility in the service state data 22 and meta-data design process by providing versioning, compatibility, flexible design process, and a standard code generation. The meta-data model based meta data, enables the tool set 12 to provide model versioning mechanisms. Figure 4 depicts a relationship model in accordance with an exemplary embodiment illustrating the relationships among the elements of the tool set 12. Specifically, the relationships between the SSDG 200, meta-data model 220, and attributes 240 are illustrated.

c. Custom templates 230 include, but are not limited to, custom attributes (e.g., user defined attributes) and default values for the state meta-data.

[0043] The tool set 12 and SSDG 200 provides a Graphical User Interface (GUI) based and/or a command line solution to generate the state data 22 of a service 20 and to create the necessary meta-data and mapping information associated with that state data 22. Moreover, use GUI features like custom dialogs with user, drag and drop of meta-data information, flexible context menus and flexible ways to create state data relationships (workflows).

[0044] The tool set 12 and SSDG 200 includes a capability to obtain user (service developer) feedbacks on meta-data generation for a service state data 22; based on the attributes 240, e.g., qualifiers, constraints and data access mechanisms associated with the state data.

a. User can give feedbacks through custom dialog boxes. This tool set 12 can query developer feedback on the type of attributes 240 e.g., qualifiers for a state data; transaction requirements, access control information, notifiable capabilities etc.

b. User can pass parameters to this generator tool set 12;

c. User can provide templates to guide the meta-data generation and mapping.

[0045] The tool set 12 and SSDG 200 is developed and configured as a pluggable framework so that it can be used as an eclipse plug in or can be included with other user interfaces frameworks. User e.g., developer for the service 20, may define an extensible set of meta-data information and generate pluggable extension mechanisms for meta-data attributes 240, e.g., data qualifiers, constraints and access mechanisms based on the service 20 needs. Provides meta-data model 220 extensibility through pluggable frameworks. This enables a service developer to support custom meta-data attributes 240.

[0046] The tool set 12 provides a set of validation facilities to validate the meta-data based on meta-data model 220 and data schema 210. Moreover, the tool set 12 also provides some runtime validation on the access mechanisms. As an example, a user can validate data access using SQL queries and CIM against the utilities supplied.

[0047] The tool set 12 enables a state workflow model for developing a service state collaboration and state relationship.

State Meta-data Processors Based on Meta Information Modeling In A Service Oriented Architecture

[0048] Disclosed herein in yet another exemplary embodiment is a method and apparatus for embedded/pluggable state meta-data processors based on a meta information model in a service-oriented architecture. This framework provides one ore more processors generated from a meta model for Gateway for Service Oriented State (GSOS) framework as described above, with an extended set of capabilities by utilizing the extensible meta-information model framework also described herein, in order to process the meta-data related to state data qualifiers, constraints and access mechanisms. These processors are defined and can be executed in a service specific controller engine, which provides process flow and orchestration capabilities.

[0049] Referring now to Figure 5, in a service oriented architecture, the Gateway for Service Oriented State (GSOS) framework 10a (Figure 2) and 10b (Figure 3), provides a flexible service state exposure model through various queries on the state of the service using the exposed state information model (e.g., state data schema). This GSOS framework also facilitates defining mechanisms to enable a service 20 to extend its state with change notification semantics. Another feature of the abovementioned frameworks 10a, 10b is the support of data transformation for the client 30 in a manner appropriate for their consumption. These extensible frameworks depend on the state data and its meta-data and a meta-data information model 220 (meta-data about meta-data). This meta-data provides state data qualifiers (security, transaction and notifiability etc), constraints (mutability, time to live) and access mechanisms (database connection, connection to CIMOM, SNMP etc). These meta-data are modeled using the meta information models 220 and this enables the application developer to create new meta models and/or extend the existing meta-models with new service defined semantics.

[0050] In yet another exemplary embodiment a framework 10c for processing meta-data and its associated semantics through embedded/pluggable and extensible meta-data processors 300 is described. Once again, it will be appreciated that the framework 10c for the meta-data processor 300 may be implemented in conjunction with the GSOS 100 and framework 10a, as well as the tool set 12 and SSDG and framwork 10b. The meta-data processsor as described herein in an exemplary embodiment may be considered embedded with the GSOS 100 or as pluggable thereto. Described herein in another exemplary embodiment is a framework 10c to use some workflow models (custom or pre-existing) to control the invocation flow and processing logics of the meta-data processors 300. In short, this permits definition of a meta information driven architecture, with platform independent models (PIM) for the state meta-data and with platform specific models (PSM) for semantic processors to process the meta-data associated with the state data.

[0051] Advantageously, the framework 10c of an exemplary embodiment includes a model driven architecture with platform independent models to define the meta-data for the state data and platform dependent models for specific (semantic) processing of this state meta-data based on the meta-data model 220. The separation of platform dependent and independent models enables an application developer to create flexible state data processing rules for any type of (EJBs, Java beans, .NET services etc) service component.

[0052] Moreover, any number of generic semantic processors may be created and utilized to manage the state meta-data processing. Such a configuration permits a service developer to write specific extensions for semantics processing. One advantage facilitated by such generic semantics processing capability is that it permits flexiblility for model extensions and application processing extensions. Another capability of the semantic processors 302 is facilitates association of work flow mechanism (custom or preexisting orchestration engines)

[0053] As shown in the Figure 2, as described above the Gateway for Service Oriented State (GSOS) is composed of a number of components. These components inlcude, but are not limited to, meta-data processors 300 (generated by meta-model tools and/or by the service developer and its extensions), interfaces to connect to the GSOS, an internal meta-data representation 102 and interfaces to connect to external systems and processors 108, 112, 116 (query, notification and transformation). In an exemplary embodiment as disclosed herein relates to meta-data processor(s) 300 and their relation to meta-data 310 , meta-data information model 220, and a processor execution flow model. Moreover, a framework 10c for service specific controller processor (a service specific meta-data processor), which is responsible for managing the meta-data processing based on service specific requirements including introspection on meta-data 310 and meta-data processor 300 flow orchestration.

[0054] The major components of this process framework 10c and meta-data processor 300 includes, but are not limited to, a meta-data information model 220, meta-data information 310, semantic information processors 302, controller meta-data processors 304, as well as interfaces 102, 104, 106 for these processors 302, 304, and syntax, semantics and work flow information 320 for these processors 302, 304.

Meta-data Information Model

[0055] Once again, meta-data information model 220 is employed to define meta-data about state meta-data. In an exemplary embodiment, this model 220 is platform independent (PI) and and platform extensible (PE) model with information including:

Meta-data syntax for state meta-data validation, versioning and consistency check;

Meta-data semantics for,

a. Common semantic information on State data constraints, qualifiers and data access mechanisms

b. Rules for processing the above meta-data semantics

c. Meta-data processing (execution of semantic logics) workf low information;

Meta-data relationship; and

Meta-data extensibility to meet service needs.

[0056] This is a flexible information model and may be expressed in any model language (e.g., XML Metadata Interchange (XMI), Meta Object Facility (MOF), Unified Modeling Language (UML) and the like) and can be used by the Service Oriented State

Data Generator SSDG 200 and tool set 12 to create a meta-data and generic semantic processors 302 to handle those meta-data 310.

[0057] As an example, we can define the semantic information on the transaction requirements on the state data through a meta-data model 220. The meta-data model 220 contains the platform independent model of the transaction attributes including transaction type, isolation level, locking etc. A service 20 developer can work with the tool set 12 as described previously herein with respect to the Service Oriented State Data Generator SSDG 200 (shown in figure 4) to generate platform specific models including a meta-data processor 300 for transaction management and transactional deployment and runtime configuration information for that specific platform.

Meta-data Information

[0058] For each service state data 22 (logical or physical mapping), there is a meta-data mapping associated with the service state data 22 with details on its constraints, qualifiers and state data access mechanisms. These meta-data 310 are instances of the meta-information model 220 and forms the core data for the GSOS 100 to work with. This mapping information enables the design of a flexible state management framework. In an exemplary embodiment, tools are utilized to create these service state meta-data 310. These meta-data are associated with a meta-model for versioning, validation, consistency and process semantics.

Semantic Information Processors Based On Meta-Model To Process Meta Data

[0059] These semantic information processors 302 are created from the platform specific modeling artifacts, which are derived from the platform independent meta model. These processors 302 provide service 20 developers an interface to define rule engines to process the semantics associated with the meta-data 310. In other words, these processors 302 enforce the constraints on the state data specified by the meta-data 310. These processors may be built in platform specific and/or container specific processors to handle meta-data constraints, qualifiers and data access mechanisms. Once example of such a processor can be a transaction processor to process the transactional requirements on a state data access. These platform specific processors 302 may be configured to work in conjunction with a platform specific transaction coordinator to execute the transactional requirements on the state-data. These processors are configured to follow some specific contracts (interfaces and process flow) to conform to the meta-data model 220 as defined earlier. These are pluggable and extensible processors based on the model version and service specific requirements.

[0060] In an exemplary embodiment, the Semantic Information Meta-data processors 302 are configured to be platform dependent where each service extensible processor 302 relates to selected meta-model semantics with a rule engine to execute the meta-data semantics. Each semantic processor 302 follows a well defined contract model dictated by th control processor 304 for interoperability between semantic processors 302. Advantageously, the semantic meta-data processsor 302 may employ a standard programming model to access and discover meta-data. In addition these processors may be employed to validate the meta-data using a meta-data model 220.

Controller Meta-data Processors

[0061] The controller processors 304 are service specific meta-data processors (acting as controller) 304, that define the contracts between the semantic processors 302 and control the flow of data between the semantic processors 302 based on the state meta data information. The service 20 developer, based on the rules defined in the meta-data and meta-data model 220 can develop these processors 304 and contracts. A tool set may be used to develop the basic skeleton of this processor 302. This processor provides a state meta-data and state data repository. This repository can be physical representation (caching) for improved performance or can be a logical representation with external storage in the service 20 or elsewhere (databases). In an exemplary embodiment, the functionalities provided by this processor 304 include, but are not limited to, validation of the meta-data with meta-data model 220, introspection on the meta-data to identify and select an appropriate processor execution engine, control of the semantic processor(s) 302 execution flow using an appropriate orchestration engine, providing contract definition among various semantic meta-data processors 302, and providing flow mechanisms between semantic meta-data processors 302. The control processor may also provide a meta-data or state data repository.

Interfaces And Process Flow

[0062] All the meta-data processors 302, 304 should follow some defined contract information and of course, would need to follow well-defined interface requirements. In an exemplary embodiment a state less model of meta-data processors is defined. These contracts are used for managing interfaces such as, life cycle management, state management, and processing contracts (data flow, validation, versioning etc). The contracts also faciltate management of process flow by coordinate work flow with a workflow engine, and implementing workflow requirements.

[0063] To reiterate, these meta-data processors 300 provides a pluggable and extensible framework 10c for managing the state data through its meta-data semantics. This state data can be local to the service or can be stored elsewhere or can be held in any external resource. These meta-data processors 300 provides a common view of the data to the client by providing abstract models (meta models) whereby hiding the internal platform specific details.

[0064] The disclosed invention can be embodied in the form of computer, controller, or processor implemented processes and apparatuses for practicing those processes. The present invention can also be embodied in the form of computer program code containing instructions embodied in tangible media 2 (Figure 1A), such as floppy diskettes, CD-ROMs, hard drives, or any other computer-readable storage medium, wherein, when the computer program code is loaded into and executed by a computer, controller, or processor 1, the computer, controller, or processor 1 becomes an apparatus for practicing the invention. The present invention may also be embodied in the form of computer program code as a data signal 3, for example, whether stored in a storage medium, loaded into and/or executed by a computer, controller, or processor 1, or transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via electromagnetic radiation, wherein, when the computer program code is loaded into and executed by a computer, the computer becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, 1 the computer program code segments configure the processor to create specific logic circuits.

[0065] It will be appreciated that the use of first and second or other similar nomenclature for denoting similar items is not intended to specify or imply any particular order unless otherwise stated.

[0066] While the invention has been described with reference to an exemplary embodiment, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from the essential scope thereof. Therefore, it is intended that the invention not be limited to the particular embodiment disclosed as the best mode contemplated for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims.